WBS 4.2-3

Knowledge Base Management Systems Study

Report #2

[redacted] The limitations contained in this legend will be considered void after September 28, 1988. T[redacted] m[redacted] reproducti~~on of this~~ [redacted]

Prepared under Contract NAS1-17555 by
Boeing Commercial Airplane Company
P.O. Box 3707
Seattle, Washington 98124

for

Langley Research Center
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Prepared by:

L S Baum
L. S. Baum

Date: 9/28/84

W. J. McClay
W. J. McClay

Date: 9/28/84

[illegible]
S. J. Lee

Date: [illegible] 28 [illegible]

Responsible Manager:

[illegible]
H. R. Johnson

Date: 9/28/84

Approved by:

[illegible]
W. A. Bryant

Date: [illegible]

TABLE OF CONTENTS

1.0 INTRODUCTION

This report represents initial findings of IPAD research into the feasibility and desirability of achieving a knowledge base management system for engineers by integrating data base and artificial intelligence technologies.

Principal issues in this regard include the following:

a) If and how various knowledge representations can be accommodated using current and/or extended data base techniques

b) how well-understood data base management techniques can be exploited in order to provide more sophistication, power and utility for a knowledge-based system tool

c) how the full capability of AI inferencing can be integrated with semantic capabilities of DBMS technology

d) how other knowledge base functions such as explanation/justification reporting and handling certainty factors can be integrated into the extended data base architecture

1.1 THE MOTIVATION FOR THE KNOWLEDGE BASE MANAGEMENT SYSTEM

Currently, AI applications are developed as isolated projects. While the same AI language or expert system building tool might be used from one application to the next, there is no attempt to make the knowledge or facts captured by one system sharable with others. There is no common repository of rules or frames or logical assertions. Also, facts which are entered into one system by the end user may duplicate facts already captured in other systems or perhaps in some data base. This causes concern not only because of the duplication of effort and wasted computer resources, but also it creates a maintenance problem as well as a consistency problem if the data is subject to change. In the future, the capability for sharing knowledge among applications will be a requirement along with clean and effective access to data in various data bases.

Knowledge sharing and more ambitious applications will result in knowledge bases that are much larger than current AI tools are able to handle. Most current tools base their search strategies on the assumption that all of the knowledge and facts reside in main memory. This assumption will face serious challenge in the future as the need for very large integrated knowledge bases come to prominence. Accordingly, search strategies will need to be much more sophisticated than is currently the case. Managing a very large number of facts such as would be contained in a large scale data base is a challenge which has not seriously been addressed by the AI community.

1.2 KBMS THROUGH INTEGRATION OF DBMS AND AI TECHNOLOGIES

There is a huge vendor and customer investment in DBMS software and applications. In the future these vendors and customers will wish to take advantage of the sophisticated inferencing capabilities provided by AI technology. And as the investment in and application of AI technology grows, there will be a corresponding push to take advantage of the sophisticated data management capabilities provided by DBMS technologies. It is the combined strength of AI and DBMS technologies which will provide knowledge base management capabilities.

Systems have been proposed which would interface an expert system to a data base management system. The expert system would be able to obtain facts from the data base and get answers to queries about data in the data base through interfacing procedures which would be invokable through clauses in rules. At best, this provides only a loose coupling with the data base, placing an unnatural division between knowledge contained in the form of rules or frames, etc. in the expert system and knowledge in the form of data semantics (e.g. relationships, constraints, and system generated actions) described in the data dictionary of the data base. This division could result in gaps or inconsistencies in knowledge and consequent inferencing. For example, how would the knowledge of the AI subsystem be aware of that of the DBMS subsystem and vice versa. For reasons of efficiency as well as for the sake of making progress towards more sophisticated tools for the management of both knowledge and data, the integrated approach seems more reasonable in that it provides a common facility which incorporates in a uniform and consistent way the strengths of both technologies.

1.3 APPROACHES TO INTEGRATION OF THE TECHNOLOGIES

Managing large quantities of data, providing views of and shared access to data, enforcement of constraints, propagated actions, concurrency control, and efficient access methods are capabilities which are currently provided by DBMS technology. This offers a solid foundation upon which to build. Indeed enhancement and application of this technology to achieve general purpose knowledge base management system technology is a very attractive approach to those vendors and customers currently invested in DBMS technology and applications.

Similiarly, a case can be made for using AI technology for the departure point for KBMS technology; an approach which might incorporate DBMS capabilities into AI products. A third approach would be to begin anew, selecting features and implementation mechanisms from both AI and DBMS technologies as appropriate. No doubt each approach will be pursued.

This paper focuses on an evolutionary approach which is based in DBMS technology. That is, it assumes DMBS technology as a base and attempts to identify functional commonality and disparity between DBMS and AI technologies, and then takes commonality into account in the integration of the two. However, many of the issues that are discussed should be considered in any effort to merge the two technologies.

Dealing with commonality may result in a common implementation of some features, or it may result in redundant, but consistent and coordinated implementation of others. A tightly coupled implementation of inferencing and query/view processing, for example, might result in a single very efficient and powerful KBMS capability. Ignoring commonality may result in duplication of effort, unnecessarily complicated user interfaces, and worst of all, gaps or inconsistencies within the knowledge processing capabilities of the system.

Another issue in the evolutionary approach to the integration of DBMS and AI technologies is that of upward compatibility. If the KBMS is to be built upon an existing DBMS, then it might be desirable to add capabilities in a way that preserves user interfaces and/or operational semantics. The issues are considered at various points in this paper.

2.0 USER INTERFACE

There are many criteria which might shape user interfaces. Some of these are the following.

- o Upward compatibility with existing DBMS interfaces.
- o Uniformity of particular classes of declarations.
- o Layering of capabilities.

In the following we use DBMS interfaces (DDL/DML data description language/data manipulation language) as a basis for discussion. In this context some of the above criteria may conflict. For example, Section 3.1 demonstrates that CODASYL set/foreign key relationships between record types may be expressed in the form of production rules. Upward compatibility with DBMS interfaces would argue for declaration of these relationships using conventional syntax in addition to syntax for declaring other types of production rules. Uniformity would argue for declaring all rules, including these relationships, using a single production rule-oriented syntax. A middle ground, would allow either style of declaration to be used. Some of these options are discussed in the following sections.

KBMS capabilities might be layered. For example one level of capability might offer selected classes of production rules, while another level of capability might offer additional rules, etc. Adding capability for handling knowledge in the popular AI representations of rules or frames need not alter the underlying DBMS philosphy that knowledge about the problem domain be expressed by means of the DDL and be maintained in the data dictionary. This is not to say that knowledge declarations must be fixed in advance and not be alterable interactively when the system is in use. Indeed, some DBMS's currently support dynamic DDL.

In some DBMS's (e.g. SQL) DDL and DML are unified - a single language in which query and view of the knowledge/data structures are practically the same. To extend such a data model to provide KBMS functionality would probably result in a user interface in which knowledge is declared and invoked in a uniform fashion.

For those who would see greater advantage to the more typical AI approach to user interface, of course it would always be possible to adopt a lamda calculus or first-order logic approach which would embody DBMS functionality. Both DBMS and AI-oriented interfaces might be offered.

3.0 KNOWLEDGE REPRESENTATIONS

In this section some of the issues regarding support for knowledge representations are discussed with respect to their incorporation in data base management systems.

One important issue is that of commonality or close ties between DBMS and AI capabilities. For example, as noted in Section 2.1, DBMS-supported relationships between records are closely related to production rules. Since real world DBMS applications may involve hundreds of relationships, this connection should not be overlooked. To do so would risk overlooking vast amounts of knowledge in current systems, or in dealing with this knowledge separately from additional knowledge specified as production rules. Section 3.1 goes even further to show that relationships can be translated to production rules. This fact has many ramifications in regard to implementation of user interface and implementation mechanisms for inferencing. For example, a single mechanism might be used to support both types of knowledge; and some of the DBMS technology for supporting relationships might be applied to implementation of the KBMS inferencing mechanism.

The correlation between frame technology and DBMS technology seems rather straight forward. On the surface, semantic network technology also seems closely related to DBMS technology. However, there seems to be some rather subtle, but significant differences. This situation gives rise to additional considerations regarding integration of DBMS and AI technologies.

3.1 PRODUCTION RULES

According to Rich [1], a production rule system consists of:

- A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule, and a right side that describes the action to be performed if the rule is applied.

- One or more databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.

- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

The left side of a rule is often called the condition side, and the right hand, the action side.

The data dictionary of a DBMS contains knowledge about data base data in the form of descriptions of relations and relationships, constraints, rules for computing virtual attributes and for propagation of user initiated modification or deletion, etc. This knowledge is supplied via statements of a data description language (DDL). Much of this knowledge can be and is for some DBMSs stated as rules. The following are examples in pseudo-DDL of rules which might be declared using actual DBMSs today:

1. IF VALUE OF PERSON-ID NOT UNIQUE THEN REJECT STORE OF PERSON RECORD.

2. IF DEPT-NO OF DEPARTMENT CHANGES THEN COPY NEW VALUE TO DEPT-NO OF EMPLOYEE FOR ALL EMPLOYEES OF THE DEPARTMENT.

3. IF DEPT-NO OF EMPLOYEE CHANGES THEN COPY NEW VALUE TO DEPT-NO OF DEPENDENT FOR ALL DEPENDENTS OF THE EMPLOYEE.

4. IF PARTS-ON-HAND < REQUIRED-NUMBER AND STANDARD-ORDER ≠ NULL THEN GENERATE ORDER.

5. IF PARTS-ON-HAND < REQUIRED-NUMBER AND STANDARD-ORDER = NULL THEN SOLICIT USER "SPECIFY NUMBER OF" parts-name "TO ORDER" AND GENERATE ORDER.

6. IF VALUE OF DEPT-NO OF EMPLOYEE REQUESTED THEN COPY VALUE FROM DEPT-NO OF EMPLOYEE'S DEPARTMENT.

7. IF VALUE OF DEPT-NO OF DEPENDENT REQUESTED THEN COPY VALUE FROM DEPT-NO OF RESPONSIBLE EMPLOYEE.

8. IF VALUE OF TOTAL-SALARY OF DEPARTMENT REQUESTED THEN CALL PROCEDURE SALARY-SUM(INDIVIDUAL-SALARY OF EMPLOYEES).

Actual DDL syntax for rules may be in the form of stand-alone statements. (This is the case for rule 1 as declared in the RIM DBMS). Or, DDL syntax for rules may be embedded as phrases in other declarations. (This is the case for rules 1 as declared in the IPIP and SQL DBMSs and for rules 2 and 3 as declared in IPIP). In some DBMSs (e.g. CODASYL [2] compliant systems) rules are specified throughout in the form (again in pseudo DDL) "ON EVENT event-specification PROCEDURE procedure-name". In RIM these declarations are called rules. In INGRESS they are called triggers. In other systems they are called assertions or ON clauses.

(It might be noted that the condition part of rules 1 through 5 are expressed in terms of events such as "if the value of DEPT-NO changes". This event-oriented syntax is sometimes used with the procedural attachment mechanism of frames (see Section 3.3). The condition part might be phrased in a more pure but equivalent, rule-oriented syntax such as "if value to be posted of DEPT-NO ≠current value of DEPT-NO". Similiarly retrieval events can be expressed in rule-oriented syntax. We leave it to the reader to made this translation).

Rule 1 above is an example of a constraint which is enforced by the DBMS to ensure that each person has a unique identification.

Rules 2 and 3 provide for propagation by the system of externally specified modifications to the data base. If for example, somebody changes the value of DEPT-NO of the sales department from 12 to 13, then rule 2 would be fired to change DEPT-NO of all employees of the sales department from 12 to 13. These modifications to the data base would repeatedly fire rule 3 which would change DEPT-NO of the dependents of these employees from 12 to 13. CODASYL specifications (1973) [3] provide for this through its ACTUAL SOURCE construct for attribute declaration. IPIP and CODASYL compliant systems provides for DDL specification of propagation of attribute modification and of record/row deletion. Chained firing of rules may occur in either case.

Rules 4 and 5 specify for an inventory data base what to do when a particular part falls below a specified minimum. REQUIRED-NUMBER and STANDARD-ORDER are attributes of the PART record. Their values may vary from part to part, or perhaps, a value for STANDARD-ORDER might not be specified (i.e., STANDARD-ORDER = NULL). So for bolt, when PARTS-ON-HAND falls below 1000, then rule 4 might be triggered to automatically initiate an order for 100 bolts. On the other hand, for fender when PARTS-ON-HAND falls below 100, rule 5 might be triggered, in which case, the user would be asked to specify how many fenders to order, and when he responds with 7, the system would generate a corresponding order. "GENERATE ORDER" might trigger execution of a procedure which prepares an order. The user solicitation might be implemented as part of the procedure. Rule 5 could be modified to advise the user to manually issue an order by changing the action part to DISPLAY "PARTS-ON-HAND FOR" part-name "BELOW REQUIRED NUMBER".

Rules 6 and 7 provide computation of the value of an attribute. If the value of DEPT-NO of an employee is requested then rule 6 would be fired. If the value of DEPT-NO of a dependent is requested, then rule 7 would be fired, which would in turn cause the firing of rule 6. CODASYL specifications (1973) [3] provide for this through its VIRTUAL SOURCE construct for attributes declaration.

Rule 8 calls data base procedure SALARY-SUM which fetches EMPLOYEE records for the department and sums over the INDIVIDUAL-SALARY attribute contained therein. CODASYL specifications (1973) provide for this through its VIRTUAL RESULT construct for attribute declaration. The ACTUAL RESULT construct can be used to specify that SALARY-SUM should be called to compute TOTAL-

SALARY of DEPARTMENT whenever an INDIVIDUAL-SALARY attribute in an EMPLOYEE record changes.

The rules capability of DBMSs can be extended to include the general rules capability supported by expert systems. Additional rules represent additional knowledge about the domain of expertise. The general capability provides for the declaration of additional classes of conditions and actions.

A CODASYL compliant DBMS provides for the declaration of entities (as records), attributes and relationships (as sets). (IPIP, which supports both the network and relational data models provides for the declaration of entities (as records or relations), attributes and relationships (as sets or foreign keys).

For purposes of upward compatibility, the DDL of such a DBMS might be extended to provide for entities, attributes, relationships, and rules. Consider the example (again in pseudo DDL) concerning parts inventory that is illustrated in Figure 3.1-1.

```
RECORD:  PART

     ATTRIBUTES:  PART-NO, PARTS-ON-HAND,
                  REQUIRED-NUMBER, STANDARD ORDER

RECORD:  ASSEMBLY

     ATTRIBUTES:  C-A-PART-NO, C-I-A-PART-NO, QUANTITY

RELATIONSHIP:  CONTAINS-ASSEMBLY

     BETWEEN PART AND ASSEMBLY RECORDS
     DETERMINED BY PART.PART-NO = ASSEMBLY.C-A-PART-NO

RELATIONSHIP:  CONTAINED-IN-ASSEMBLY
     BETWEEN PART AND ASSEMBLY RECORDS
     DETERMINED BY PART.PART-NO = ASSEMBLY.C-I-A-PART-NO

RULE:  CONTAINS-PART

     IF P1 CONTAINS-ASSEMBLY A1
          AND P2 CONTAINED-IN-ASSEMBLY A1
          THEN P1 CONTAINS-PART P2

RULE:  CONTAINED-IN-PART

     IF THE P1 CONTAINS-PART P2
          THEN P2 CONTAINED-IN-PART P1
```

Figure 3.1-1 - Parts Explosion Schema Containing Declarations of Records Attributes, Relationships, and Rules

This is the classical network representation of the parts explosion application, and is illustrated in the data structure diagram of Figure 3.1-2.

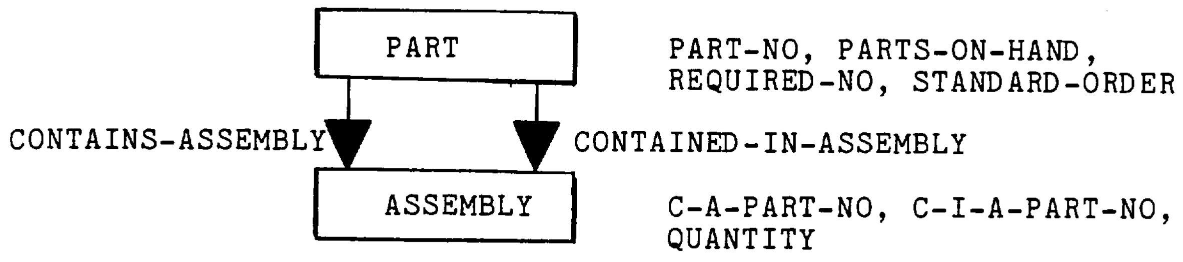

PART
PART-NO, PARTS-ON-HAND, REQUIRED-NO, STANDARD-ORDER
CONTAINS-ASSEMBLY
CONTAINED-IN-ASSEMBLY
ASSEMBLY
C-A-PART-NO, C-I-A-PART-NO, QUANTITY

Figure 3.1-2 Data Structure for Parts Explosion

This model provides for a many-to-many relationship between parts in that:

- One part may contain many parts
- One part may be contained in many parts

The QUANTITY attribute of assembly notes how many subparts are contained in a superpart.

The record-relationship occurrence diagram of Figure 3.1-3 illustrates the data structure as applied to the part wheel and some of the parts contained in it.

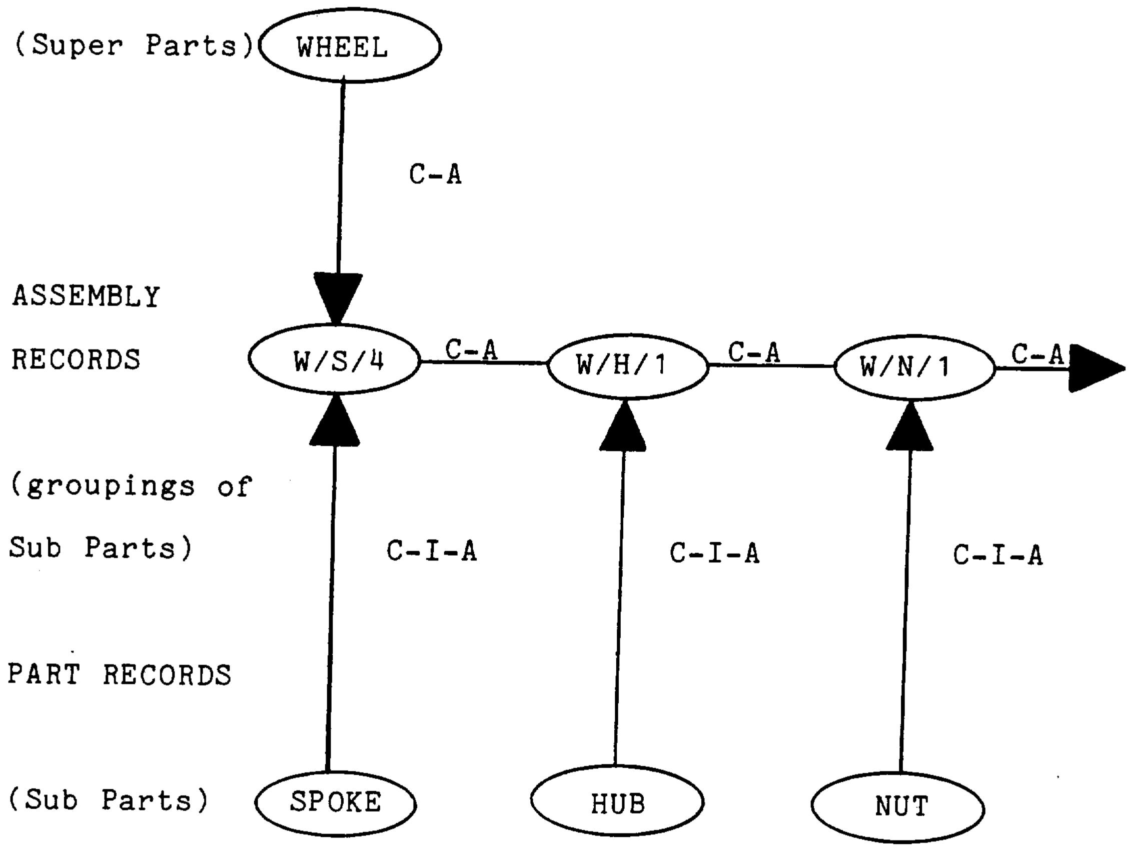


Figure 3.1-3 PART RECORDS

This wheel (as represented by a PART record) contains the three assemblies shown and therefore 4 spokes, 1 hub, and 1 nut. A different kind of wheel (as represented by another PART record) might contain a different number of spokes, hubs, or nuts (as represented by the same PART records but linked to the second wheel by three additional ASSEMBLY records).

Both the relationships and the rules of Figure 3.1-1 express knowledge about parts and assemblies. It is important that the mechanisms which draw inferences over the one type of knowledge are aware of the other, and vice versa. Otherwise there may be knowledge gaps or inconsistencies.

To illustrate this point, consider that the two relationships could be expressed as rules as follows:

```
RULE:  CONTAINS-ASSEMBLY
     IF P.PART-NO = ASSEMBLY.C-A-PART-NO
          THEN P CONTAINS-ASSEMBLY A

RULE:  CONTAINED-IN-ASSEMBLY
     IF P.PART-NO = ASSEMBLY.C-I-A-PART-NO
          THEN P CONTAINED-IN-ASSEMBLY A
```

It would be easy to overlook this commonality in the two representations of knowledge.

The commonality between relationships and rules as demonstrated in the preceeding allows for a common encoding of the two types of declaration through compilation so that an inference engine from an existing expert system (as modified to access the data base) might be used to implement the KBMS inference mechanism.

Another implication of this commonality is that for the sake of uniformity all rules, including those supported by current DBMS technology, could be expressed using the production-rule syntactical form. The preceding paragraphs suggest how relationships might be expressed. An earlier aside suggested how events such as update and retrieval could be expressed on the condition side of a rule. This capability would require visibility of registers containing values to be posted and type of command to be executed.

A middle course for user interface would be to continue to support existing DDL, syntactical constructs, and to support as well corresponding declarations as rules.

For example, production rules might be stored upon compilation in the data dictionary using one or more of the access methods (e.g. hash, index, index-sequential) provided by the DBMS (assuming that the data dictionary, as is not uncommon, is itself implemented as a data base).

Particular access methods might be specified for particular classes of rules depending on how frequently these rules are invoked and required response times for inferencing. (Some DBMS's support main memory-oriented access methods for very high performance. Such an access method could be added to a DBMS if necessary to support very fast inferencing). Traditional DBMS access methods are likely to be very useful for retrieval of knowledge for future expert systems which must accommodate very large numbers of rules. Use of DBMS facilities for storing knowledge should facilitate knowledge management capabilities paralleling traditional DBMS capabilities such as concurrency control, views, etc.

There a variety of physical data structures (e.g. embedded pointers, pointer arrays, indexes) used to implement occurrences of relationships on the data base. Sometimes a DBMS offers more than one of these, so that efficient processing of relationships can be achieved for various access patterns. The correspondence between relationships and rules suggests that these data structures would be applicable to supporting at least a subclass of production rules to achieve efficient inferencing for various reasoning patterns.

An important issue in production rule systems is how to efficiently identify at any given time those rules which are candidates for firing. DBMS technology has evolved efficient mechanisms for rule identification for the class of rules that it supports. These mechanisms might be used in KBMS implementation. These mechanisms include embedding rule identifiers in the data dictionary description of records, attributes, and relationships. Use of these mechanisms would involve tight integration of the implementations of AI and DBMS capabilities.The KBMS inference mechanism might be tightly integrated with the query/view processing mechanism of the DBMS. (Query and view processing are nearly equivalent in some DBMSs). The query optimizer of some DBMSs decomposes a complex query into a tree of subqueries. Execution of these subqueries may be sequenced from the root to the leaves or vice versa depending on the query and the description and the population of the data base. Consider, for example the following query:

WHICH PARTS ARE CONTAINED IN WHEEL.

Suppose that only the CONTAINS-ASSEMBLY and CONTAINED-IN-ASSEMBLY rules declared above are available. Then the query could be satisfied by applying first the rule (subquery) CONTAINS-ASSEMBLY to wheel and the rule (subquery) CONTAINED-IN-ASSEMBLY to the resulting assemblies or it could be satisfied by applying first CONTAINED-IN-ASSEMBLY to a part (spoke or hub or nut or fender or axel, etc.) and then applying CONTAINS-ASSEMBLY to the resulting assemblies. These strategies for execution of the query are closely related to if not actual examples of the production rule concepts of forward and backward chaining. Some DBMSs provide this limited "forward and backward chaining" mechanism today. The mechanisms used by these DBMSs could be incorporated in the implementation of a general inferencing mechanism and/or the general implementation could be used to greatly enhance current query optimization technology. (The corresponding term for a KBMS might well be inference optimization).

Knowledge declared as DBMS relationships becomes more involved as constraints on these relationships are taken into account. Expression of these constraints as rules would involve augmenting the conditional and action sides of those rules corresponding to relationships and/or specifying additional rules.

IPIP supports the propagation of value modification for attributes and record deletion via relationships. Thus modification of the value of a PART-NO of PART (see Figure 3.1-1) would result in system-generated modification of C-A-PART-NO in numerous ASSEMBLY records via the CONTAINS-ASSEMBLY relationship. This modification might propagate back via the CONTAINS-ASSEMBLY relationship to PART-NO in other PART records, and then back to C-A-PART-NO in ASSEMBLY records, etc. (The IPIP value modification capability is a generalization of CODASYL's). Or the attribute modification might be propagated through other relationships in a fashion analgous to rules 2 and 3 above. Similiarly deletion of a single record could propagate in complex ways to various records which are directly or indirectly related to it.

The particular path of migration, and hence the sequence in which rules are fired, depend on whether attributes which determine relationships are null or have a value.

There may be many rules corresponding to DBMS-declared constraints other than those on relationships.

DBMS applications may involve hundreds of record types and relationships. Multiple constraints may be declared for a single record type or relationship, and multiple relationships may be declared between a pair or record types. Thus current DBMS technology does accommodate the equivalent of rather large bases of rules which may be fired in a myriad of patterns. Issues concerning efficient rule retrieval and in the optimization of the sequencing of rule firing (e.g. forward and backward chaining) are nontrivial even for existing DBMS technology.

A general rule capability for a KBMS should provide for reference to both data base and non data base information. This should include as noted above reference to registers containing values to be posted and indicators specifying the type of operation to be performed (update, retrieval, etc.). A general rule capability should provide for specification complex boolean conditions and for composition of rules.

CODASYL compliant DDL does not allow, for example, rules such as CONTAINS-PARTS which are the composition of other rules (CONTAINS-ASSEMBLY and CONTAINED-IN-ASSEMBLY in this case). There are some instances of rule composition in DBMS technology. A general capability should take these into account. For example, a nested SQL query can specify nested join operations. This corresponds to a rule which composes rules corresponding to the relationships discussed above. And, IPIP provides the STRUCTURE construct which defines an aggregation of possibly many records which may be related by many relationships. A record or relation may be mapped to a structure. Update or retrieval of such a "structure-defined" record results in the traversal of many relationships. When these relationships are considered in terms of corresponding rules, then a structure declaration corresponds to a rule over one or more other rules, and execution of a rule corresponds to the firing of these underlying rules. Firing of the rules corresponding to these relationships are conditional based on the null or values state of attributes which govern the relationship. A CASE phrase (not currently implemented) provides for conditional firing based on conditions on values of these attributes.

Figure 3.1-4 illustrates the declaration of two structures. (Note that the declaration of one incorporates the declaration of the other). Figure 3.1-5 illustrates an occurrence of the structure as determined by an occurrence of the root record. Figure 3.1-6 illustrates the mapping between a structure-defined record and corresponding records within the structure.

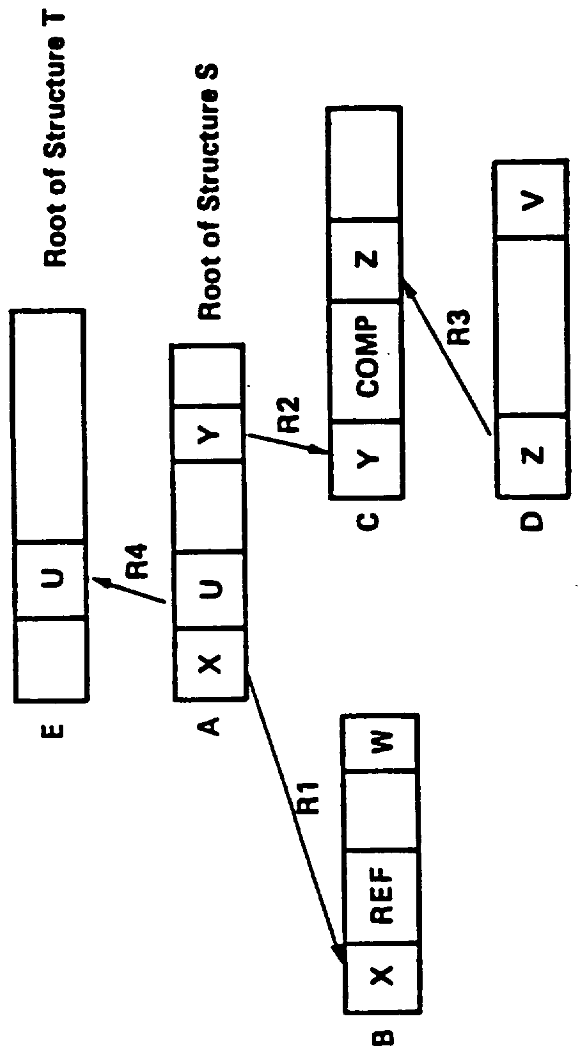
Certainty factors are very common in Production Rule expert systems and should be accommodated by the knowledge declaration capability of the KBMS. Within the definition of a production rule the certainty factor could be specified in the typical -1 to +1 range of values (see Figure 3.1-7). Range checking would automatically be applied to these values as rules were entered into the system. Only the right side (THEN portion) of the rule can have a certainty factor and if none is specified then certainty is assumed to be +1.0 which is interpreted as absolute certainty that the conclusion is true. Negative numbers indicate the degree of certainty that the conclusion is false. In either case the computations surrounding these rules are as follows:

1. The IF portion is first calculated by applying AND operators then OR's where certainty of X AND Y is simply MIN (X, Y) and the certainty of X OR Y is MAX (X, Y).

2. The certainty of each conclusion is the THEN portion is calculated by multiplying the certainty of the conditions (as found in 1) by the conclusion's certainty.

The conclusions along with their associated certainties can be used to infer other conclusions but when the certainty of a conclusion is at or near zero then it is dropped since zero is interpreted as "I don't know."

Figure 3.1-7 illustrates a production sysem incorporating certainty factors in its rules.



```
STRUCTURE S
  ROOT A
  THRU R1 TO B OCCURRENCE REF
  END
  THRU R2 TO C OCCURRENCE COMP
      THRU R3 TO D
      END
  END
```

```
STRUCTURE T
  ROOT E
  THRU R4 TO STRUCTURE S
  END
```

Figure 3.1-4 Declarations of Structures and of Structure T in Terms of Structures

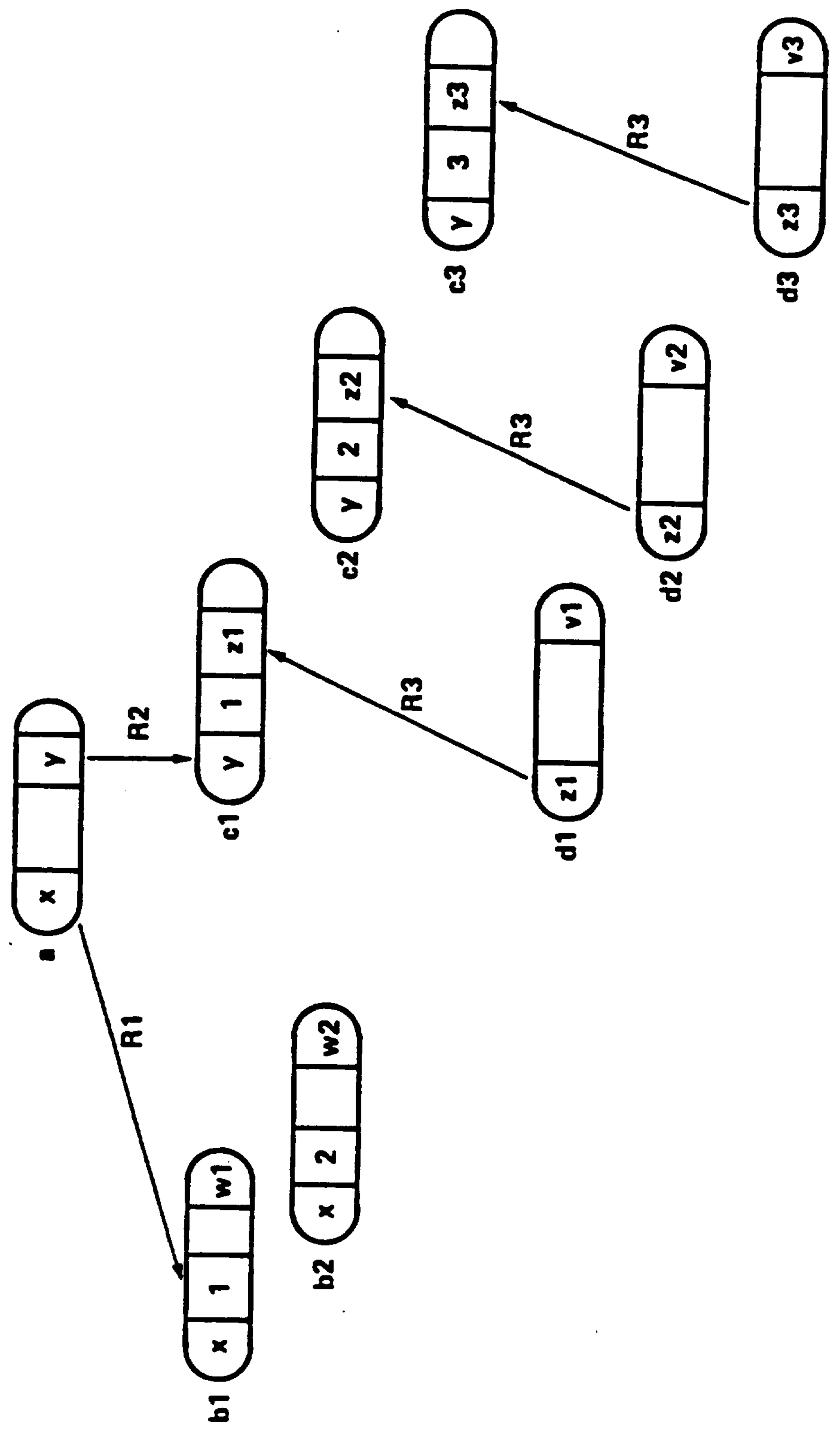


Figure 3.1-5 An Occurence of Structures

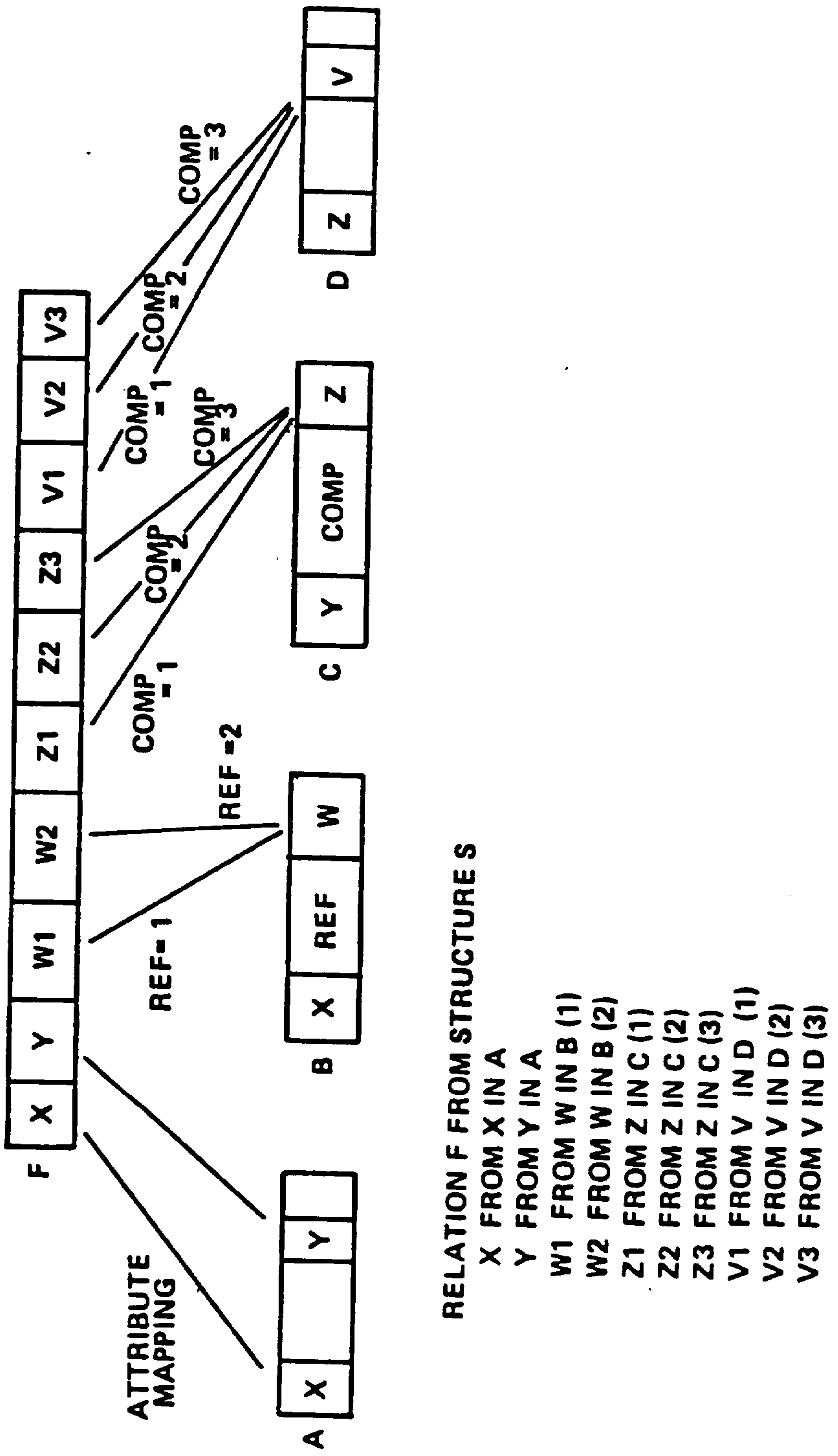


Figure 3.1-6 Mapping Entity Type F from Structures

```
IF        EMPLOYEE.  MARITAL-STATUS = MARRIED
AND       EMPLOYEE.  YEARS-OF-SERVICE > 3
THEN      STABLE (+0.6)

IF        EMPLOYEE.  YEARS-OF-SERVICE > 20
THEN      STABLE (+0.9)

IF        EMPLOYEE.  YEARS-OF-SERVICE > 1
THEN      STABLE (-0.5)

IF        EMPLOYEE.  STABLE
AND       EMPLOYEE.  AGE > 40
THEN      MANAGEMENT-MATERIAL (+0.6)

IF        EMPLOYEE.  YEARS-OF-SERVICE > 10
OR        EMPLOYEE.  YEARS-OF-SERVICE > 16
THEN      TECHNICAL (+0.9)

IF        MANAGEMENT-MATERIAL
OR        STABLE
AND       EMPLOYEE.  YEARS-OF-SERVICE > 30
OR        TECHNICAL
THEN      RETENTION-STATUS
```

Figure 3.1-7 - Production System Incorporating Certainty Factors

3.2 SEMANTIC NETS

A semantic network has the form of a directed graph composed of nodes (representing objects, concepts or situations) and arcs (representing the relationships between these entities). A simple example of a semantic net is given by:

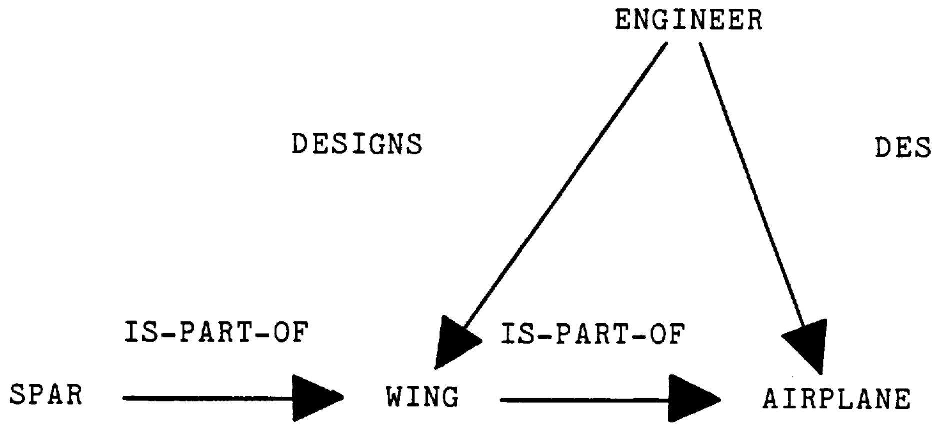


Figure 3.2-1

One approach to implementing semantic networks in CODASYL-based systems is to map nodes and arcs in a network to schema declarations of record types and set types on a one-to-one basis, respectively. This straight forward mapping of semantic networks to schema declarations requires multiple declarations of sets with the same name (e.g. IS-PART-OF in the above example), which is not permissable.

One might take a different approach using the parts explosion schema of Figure 3.1-1 (see also Figure 3.1-2). In this approach, spars, wings, and airplanes would be represented by occurrences of the PART record. This gives rise to the question of whether instances of PART record represent abstract concepts (e.g. spar, wing and record) or instances of these (e.g., 727, 737, 747), or whether instances of PART record used for both purposes.

The attributes for wing as an abstract object are different from those particular instantiations of it. For example PARTS-ON-HAND is relevant to actual parts, but not to parts in the abstract. Indeed, an IDEAL-PART record might contain only a single field which contains values such as the text string "WING". So it seems that a single PART record type will not suffice for both purposes. Let us then introduce an IDEAL-PART record type, occurrences of which represent parts in the abstract. This could be done by mirroring the parts explosion declarations schema of Figure 3.1-2 to have record types IDEAL-PART and IDEAL-ASSEMBLY and set types IDEAL-CONTAINS-ASSEMBLY and IDEAL-CONTAINED-IN-ASSEMBLY (see Figure 3.2-2). These declarations provide for modeling relationships between parts in the abstract. The resulting network of record occurrences on the database would, however, include assembly records which have no corresponding nodes in the original semantic network.

It is not clear, though, whether the assembly records are extraneous to the modeling problem at hand, or whether the semantic network has failed to model part of the problem. The QUANTITY attribute of the IDEAL-ASSEMBLY record specifies how many subparts of a particular part are contained in a super part (e.g. 4 bolts in a wheel, 6 bolts in a cam). The semantic network does not seem to accommodate this knowledge.

Proceeding with the PART/IDEAL-PART, ASSEMBLY/IDEAL-ASSEMBLY schema, it remains then to associate parts in the abstract with actual parts. To do this an IS-A set with owner IDEAL-PART and member PART is declared (see Figure 3.2-2). The IS-A relationship is based on a part type identifier included as attributes in both record types. Thus an occurrence of the IS-A set would relate an abstract part (e.g. airplane) to each instance of it (e.g. 727, 737, 747).

It seems that the inference engine would have to distinguish between records representing abstract and actual parts. The DDL would have to be extended to provide for the declaration supporting this. The CODASYL record declaration is a typing construct; that is it represents a collection of like objects. The IDEAL-PART record also plays a typing role in this approach. The difference between these typing functions bears further investigation.

The need for IS-A relationships between other IDEAL-RECORD/RECORD pairs will occur in this approach. This leads again (with the strategy of one-to-one mapping of arcs to sets) to multiple sets with the same name which is not permissable. The alternative is to realize that IS-A is a type of relationship between nodes, and not the name of arcs in the semantic network. Constructs could be added to CODASYL set syntax for typing (e.g., IS-A, KIND-OF, etc.). Arcs in the network would be mapped to uniquely named sets which would be subtyped as in the network.

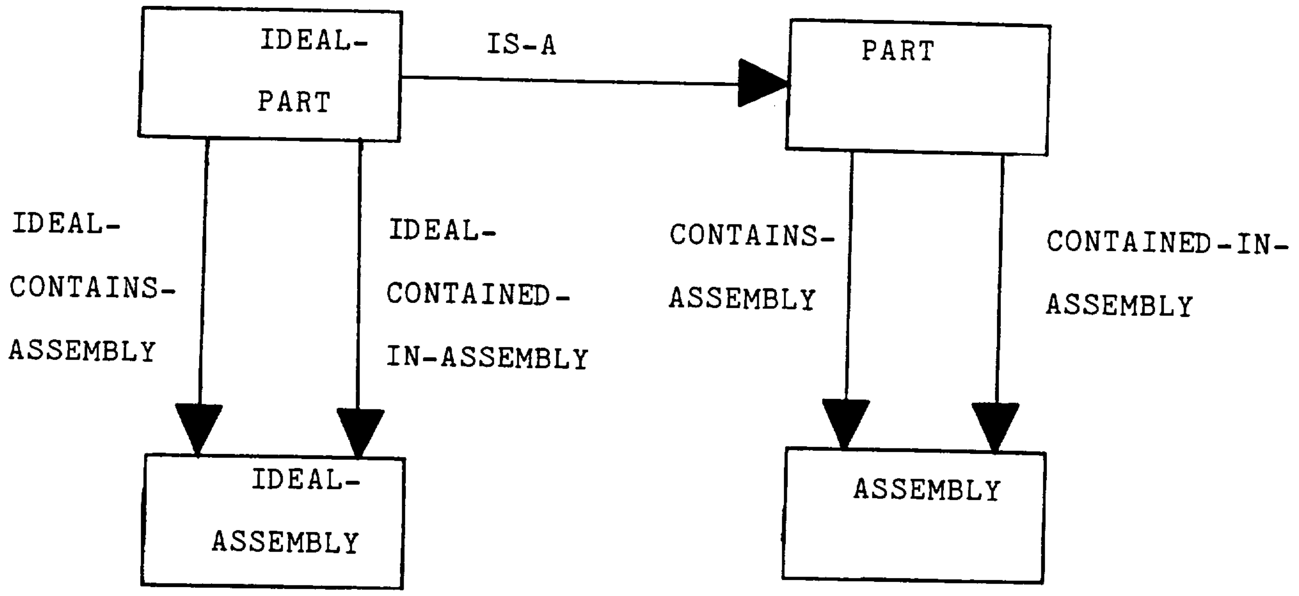


Figure 3.2-2 CODASYL Data Structure Diagram for Realizing Semantic Network of Figure 3.2-1

Implementation of semantic networks via the relational data model brings up the same fundamental issues of representing parts in the abstract, actual parts, and the linkage and distinction between the two.

The rules for inferencing over a semantic network depend upon the nature of the relations (arcs) that are in the net. In addition, it is important that the system understand the proper rules for composition of relationships; i.e., given

```
A ----------> B ----------> C
```

How is A related to C? For example, we might want the IS-PART-OF relationship to be transitive and for the system to be able to infer that SPAR is also part of AIRPLANE:

$$\text{SPAR} \xrightarrow{\text{IS-PART-OF}} \text{WING} \xrightarrow{\text{IS-PART-OF}} \text{AIRPLANE} \implies$$

$$\text{SPAR} \xrightarrow{\text{IS-PART-OF}} \text{AIRPLANE}$$

On the other hand, given

$$\text{John} \xrightarrow{\text{FATHER}} \text{Mike} \xrightarrow{\text{FATHER}} \text{Susan}$$

we certainly do not want the system to treat this relation transitively, in fact we would probably want to give the system an inference rule that would result in the system deducing that John is Susan's grandfather.

Similarly, given

$$\text{Mark} \xrightarrow{\text{BROTHER}} \text{Mike} \xrightarrow{\text{FATHER}} \text{Susan}$$

We would want the system to infer that Mark is Susan's uncle.There is no commonly accepted method in semantic network technology for specifying how to compose relationships. Such rules are required for an operational system, and if semantic networks are to be implemented in terms of DBMS construct as outlined above, then the DDL must be extended to support the declaration of these rules.

It should be noted that the above approaches to implementing semantic networks in terms of DBMS constructs represent initial research. The specific approach discussed above has not been thoroughly checked out for completeness and consistency. Efficiency aspects of that approach have not been investigated thoroughly, but mirror accessing of PART/ASSEMBLY and IDEAL-PART/IDEAL-ASSEMBLY records to deal with actual parts and knowledge about them seems to pose very serious performance problems.

Another line of investigation that should be pursued is that of implementing DBMS constructs (e.g. records and sets) in terms of semantic networks. This should provide additional insights as to the integration of the two technologies, and may even lead to refinements of the approaches discussed above.

3.3 FRAMES

In simple terms, a frame can be viewed as a sterotypical representation of any object concept. A concept is represented as a data structure with a concept name, slots, links, values, etc., much the way a record description is declared in a data base schema. Frames are typically more generalized structures than data base record structures but have many similarities. An example of a chair frame may appear as follows:

```
name:   CHAIR
links:  a-type-of FURNITURE
slots:  number-of-legs (default 4)
        material (default WOOD)
```

A rough correlation betwen frame and data base terminologies can be made as follows:

FRAMES	DBMS
Concept/Frame	Record type/Relation
Frame instantiation	Record occurence/Row
Link	Inter-record relationship for generalization
Slot	Attribute
(can contain)	(can contain)
value/null	value/null
concept name	--
concept instantiation	inter-record relationship for aggregation
procedure	--
reference to procedure	reference to procedure

Each of the DBMS capabilities seems to be a proper subset of the coresponding frame capability. For example, data base procedures are usually associated with attributes for all occurences of a record type, whereas in frames, procedures are attached on an instantiation basis. Thus the same procedures would be associated with attribute X for all occurrences of record A, for example, but might vary for slot Y for instantiations of frame B.

Similarly, attributes are normally associated with all occurrences of a record type (though they may be valued or null), whereas a particular slot may or may not be associated with an instantiation of a frame. Frame instantiations may therefore vary radically in structure from each other, whereas record occurrences (at least at the logical level) are uniform in structure.

Another difference is the generality with which one frame can be related to other frames. One way to accomplish this, is by stating a link type in the frame concept declaration, as in the example where a CHAIR is a-type-of FURNITURE. In this case all instantiations of the CHAIR frame will inherit the properties of the FURNITURE frame. Another way is by virtue of a value assigned to a slot in a frame instantiation such as, the material of the chair is WOOD. Here, each instantiation may or may not inherit the properties of some other frame by virtue of the value assigned to the slot. Still another method would be to relate one instantiation of a frame to some other instantiation. This could be used to express the relationships in the parts explosion example in figure 3.1-1. These are equivalent to inter-record relationships used for aggregation in DBMS.

DBMS technology, however, exceeds frames technology in areas such as data/sharing (in the sense of concurrency control) and views of data.

DBMS technology such as multiple access methods and concurrency control can probably be applied to frames technology at least to support that subset which corresponds to DBMS modeling capabilities. On the other hand, frames technology could be used to extend DBMS technology.

3.4 DEMONS AND PROCEDURAL ATTACHMENT

Demons and triggers, termed procedural attachment, provide immediate execution of object code when a specific condition is reached in the knowledge base. Procedural attachment are as used in frames described in section 2.3 FRAMES. Procedural attachment may be used with any knowledge representation formalism. However, if used with well defined mathematical logics like relational or predicate calculus, it may invalidate their precise definitions.

Procedural attachment could be developed from the DBMS concepts of computed attributes and propogated actions. See the pseudo-DDL DBMS rules in section 2.1 PRODUCTION RULES. Rule 4 triggers the generation of an order when parts-on-hand fall below the REQUIRED NUMBER. The knowledge base applications require the same capability. The type of triggers in a knowledge base are for supplying defaults, starting procedures when a value is modified, or when a value is placed or needed. The function of these procedures would vary from value computation to searching association links of the knowledge base for an inherited value. Two extensions to the DBMS are required. First an interactive procedure definition capability which will allow both the knowledge application designer and the application to generate object code. Secondly the object of the procedure must be present in computer memory when the procedure is needed.

3.5 MULTIPLE KNOWLEDGE REPRESENTATIONS

In order for a general purpose KBMS to serve a wide user base, it should be capable of handling as many different knowledge representations as possible. Ideally, the user should be able to specify frames, semantic nets, production rules, first-order logic, or even some mixture of these according to the needs of each subdomain of the problem. Then when knowledge is declared by the user, the KBMS would select the appropriate representation for its implementation.

Perhaps the most difficult problem associated with supporting multiple, interacting knowledge representations is the creation of an inference mechanism which could handle a nonhomogeneous knowledge base. This problem is analogous in some respects to the support of multiple physical structures by present day DBMS's. In this case the DBMS must be able to search various types of data structures, and extract and integrate data to execute a single query. Just as the modern DBMS insulates .the user from access methods, the KBMS should insulate the end user from knowledge representations. Just as data administrators are responsible for specifying access methods, so should knowledge administrators be charged with associated knowledge domains with knowledge representations.

One approach to supporting multiple knowledge representations would be to develop a common underlying knowledge representation and to develop mappings between it and the various other representations. This approach bears some resemblance to the natural language understanding problem which attempts to find an internal representation for the underlying meaning of a sentence or paragraph. For much the same reasons as we want to find a universal representation for knowledge. A leading natural language understanding researcher, Roger Schank, has found ways to map complex context dependent semantic relationships into a form known as conceptual dependency. This process only operates on restricted problem domains using a knowledge based system to perform the mapping.

4.0 PERSPECTIVES OF KNOWLEDGE

Just as DBMS technology supports views of a data base, KBMS technology should support perspectives of a knowledge base. The usual DBMS capabilities for creating views (subschemas) of data (facts) should be subsumed by KBMS technology. Thus for example, a perspective for the schema of Figure 3.1-1 might contain only a projection (selected attributes) of the PART record, leaving out the ASSEMBLY record. This might be done for simplification of the user's view or for security reasons.

Perspectives should provide views of knowlege as well as of facts. So in the PARTS example, the perspective should provide rules (knowledge) dealing with PART, but not rules dealing with ASSEMBLY, since the user is not aware of the latter. Explanations for users of a perspective should be phrased in terms of the rules of the perspective, rather than those of the underlying schema. An explanation in terms of the underlying rules (knowledge) would be no explanation at all for the user of the perspective.

Rules (knowledge) of a perspective must be derived from and be consistent with underlying rules. Conversely, certain knowledge declared in the schema which is applicable to facts represented in the perspective must be represented in the perspective. Anything less will leave the user uninformed with respect to the essential semantics of his perspective of the knowledge base.

Some underlying rules (knowledge) might not be essential, though applicable, to facts in the view. In general, rules necessary to understand update are essential, while rules pertaining to retrieval and reflective deductions are not. For example, a perspective might make available facts regarding employee salary, years of service, etc. but not make available underlying rules regarding salary adjustment, promotion and retention policies based on these facts. On the other hand, if the view provides for update of employee salary, the rules concerning constraints on salary must be provided.

CODASYL subschemas provide for the inclusion of relationship delcarations in a subschema. As noted in Section 3.1, this amounts to inclusion of rules in the subschemas.

Consider the mapping in Figure 3.1-6 between record F and the underlying structures. In this case, rules should be included in the perspective to reflect the knowledge represented by the relationships in the structure (see Figure 3.1-4), Rules in the perspective would be formulated in terms of attributes of record F, and would document update interdependencies between these attributes.

Bibliography

1. Rich, Elaine, Artificial Intelligence, McGraw-Hill, 1983.

2. CODASYL Data Description Language Committee Journal of Development, January 1978.

3. CODASYL Data Description Language Committee Journal of Development, June 1983.